

RALPH E. JOHNSON

Assistant Professor  
Department of Computer Science  
University of Illinois at Urbana-Champaign

## A Consumer's Guide to User Interface Design

### ABSTRACT

A computer system's user interface has a big impact on its acceptability and usefulness. There has been a lot of attention given to making software more user-friendly, but this has often generated more heat than light. This paper outlines the principles of good user interface design and discusses how to tell whether they are being followed or not.

### INTRODUCTION

Software companies seem to have discovered the importance of good user interfaces. Every software package is promoted as being "user friendly," but many programs are still hard to use and easily confuse the user. A good user interface is more than windows, menus, or mice. Designing one takes careful attention to the needs of the user and to the ability of people to learn new ideas and predict the behavior of complex systems.

Software engineering education has traditionally placed little emphasis on user interface design, so many software designers ignore the user interface. Even when designers realize the importance of a good user interface, they may not be able to afford to design it properly. Moreover, there is no such thing as the perfect user interface, since an interface must be designed for a particular class of users. Any user interface will be ill-suited for some user. Thus, it is important for the buyer of a software package to be able to evaluate its user interface.

The commercial success of machines like the Macintosh has con-

vinced software designers of the importance of a good user interface, and several books have been written recently that teach software engineers the principles of user interface design. Some of them are *The Elements of Friendly Software Design* (Heckel, 1989), a sermon on user interface design, *Designing the User Interface* (Schneiderman, 1986), which teaches human factors to software engineers, and *Designing User Interfaces for Software* (Dumas, 1988), which contains an extensive list of guidelines for software engineers to follow. This paper is aimed at the consumer of software, not the designer, and especially at librarians.

## PRINCIPLES OF GOOD USER INTERFACE DESIGN

The most important word in "computer user interface" is user. Unless the designers of a computer system understand the needs and capabilities of the users, the system will not be easy to use. Whether a program features pull-down menus or function keys is not nearly as important as whether it was designed to be easy to learn by novices or to be efficient to operate by experts. Thus, a program's user interface must always be evaluated from the point of view of a particular group of users. However, as long as all of a computer system's users are human, they will have many characteristics in common.

The principles of good user interface design can be divided into three categories: (1) have a good idea; (2) put the user in control and (3) take the user's point of view. Unless users can understand the purpose and structure of an application, trying to make a good user interface for it is like trying to whitewash a garbage dump. Putting the user in control means making the application understandable and predictable, so that people can use it confidently and accurately. Taking the user's point of view means adapting the user's vocabulary, making the same assumptions that the user makes, and understanding the needs and characteristics of the user. The first and second sets of principles suggest ways of designing a user interface that make it better for all users, while the third relies on knowing details about the particular set of users.

### Have a Good Idea

User interface design is a form of communication. Its purpose is to communicate the functions of the application program to the user. A pointless story well told is not satisfying, and a program that is poorly designed cannot be salvaged by a fancy interface. Even if the program addresses a real need, the solutions it offers might be too complex or solve the wrong part of the problem.

### *Keep It Simple*

It is important for computer systems to be as simple as possible. Simple systems are easier to learn and understand, and users will make less mistakes and be more productive once they understand an application. While it is important for an application to have all the features needed to solve a complete set of problems, software engineers more often err on the side of providing too many features. The result is unreliable and unmanageable systems.

Software engineers always have a model of a program that consists of the kinds of data that the program uses and the ways in which that data can be manipulated. A simple design is one that minimizes the kinds of data and the kinds of operations on that data. A design is often made simpler by making the same operations apply to many different kinds of data. For example, there can be a single "print" operation instead of "print invoice," "print budget," and "print circulation report" commands. A design can also be made simpler by breaking complex operations into simpler ones. It is fine to let the system perform many steps at a time, but the user should be able to think in terms of the simple steps in order to predict what the program will do.

It is easy to be impressed by a complicated program with many features. However, simple designs often take a lot of work. Many complex systems have no overall design but are the result of adding feature after feature to an increasingly unstructured system. One should not be led astray by lists of features, but insist upon well thought out designs that integrate solutions to all the relevant problems.

Although a system may seem simple to the user, it need not have a simple internal organization. For example, users can add or delete text with ease using a word processor. However, the algorithms to format text might be complex. The user does not care about the complexity of the algorithms, only the results of adding or deleting text. As long as these results are easy to predict, the system seems simple.

### *Copy Reality*

Good designs are often modeled after physical processes that users already understand. Most data processing systems copy the paper system that they replace, so their designs borrow the years of experience that went into the designs of the paper systems. A good example is the card catalog. Most library patrons understand (or think they understand) how the card catalog works, so it is not surprising that most computerized systems for assisting patrons are modeled after the card catalog. Doing so builds upon the users' existing knowledge.

Many of the problems for which applications are now being built have no such system to build upon, so it is essential that their designers

come up with good metaphors to follow. A good example are the hypertext-based scholar's workstations (Meyrowitz, 1986). These systems integrate text and graphics and allow the user to browse through documents in many ways, adding comments and references to other documents. Designers try to manage the problems arising from letting many people change a single document by borrowing ideas from newspaper or magazine editing, but none of the solutions seem quite right. The major problem is that nobody has ever done anything like this before, so there are no solutions to reuse.

Given a good idea, a simple design is achieved by consistency of purpose, eliminating everything that goes against the central theme. This contributes to the illusion presented by the computer. Obvious mistakes in implementation destroy the illusion, so software bugs prevent a user interface from communicating the purpose of the application.

### **Put the User in Control**

A user who is confused or who does not know what a program will do next is not in control. Users need an accurate mental model of a program to be able to predict what it will do. Programs whose form and function follow a simple model are easiest to learn and understand. Thus, it is very important that a program has a simple underlying model (the good idea) and that the user interface consistently expresses that model. Consistency is absolutely essential in making a program easy to understand, since it lets users predict the behavior of parts of the system that they have not seen from the parts that they have seen.

### *Providing Feedback*

Users are most likely to learn the model if the application provides feedback through status messages and error messages. People learn best by having their mistakes corrected immediately. If a long period of time occurs between a user's command and the system's response, it may be difficult to figure out the reason for the response. Immediate feedback makes the user feel like he or she knows what is going on, so it enhances the feeling of control. When the computer pauses for a long time, the user is likely to wonder if the computer is broken. However, immediate feedback also improves the actual control that the user has over the application, since the user will be able to spot errors sooner and correct them faster.

### *Online Documentation*

All users, even the most experienced, will sometimes need more information about how to use an application. Online help and docu-

mentation lets a user experiment with a system and learn about new parts of it. The user should be able to get help at any point in the application. This will eliminate the possibility of getting hopelessly confused and lost. Online documentation is not a replacement for printed documentation, since printed documentation is better as a tutorial and for learning about functions that have not been used before. However, a user should be able to complete most work without having to consult written documentation.

Online documentation is often not as well written as it should be. Since it is used more than the paper documentation, its quality should be at least as high.

#### *Minimize Memorization*

People are much better at recognizing detailed information than they are at recalling it. Users should be able to select commands and options from lists and so use their ability to recognize correct choices. This is the primary advantage of menus over commands to control an application.

Screen layouts should be designed to minimize memorization. If the user needs to combine two types of information to make a decision, it should be possible to display them both at the same time. One sign that this principle is violated is if users are forced to repeatedly flip between screens to compare results.

Prompts for data entry should describe any special formats that are required. For example, a prompt for a date should describe whether a purely numeric (mm/dd/yy) or an alphanumeric (dd MMM yy) entry is required. Computers are much less flexible than people at adapting to different formats, so it is only fair that the program remind the user of its particular requirements.

Online help and good status messages are very useful at jogging the user's memory. Each screen should be labeled so the users always know where they are.

#### *Consistency*

Consistency is extremely important in reducing the need to memorize. The same command should mean the same thing in every part of the application. Systems like UNIX that are a collection of different programs are often very inconsistent in their commands, since these programs have many different authors (Norman, 1981). Depending on the program, the command to leave a UNIX program might be *x* (exit), *q* (quit), *bye*, or *^D* (the end of file marker). It is no wonder that new users find UNIX confusing.

A consistent vocabulary will reduce the number of new concepts

that the user must learn. A computer neophyte is likely to get confused if told to *press* a key, then to *type* a key, then to *hit* a key. All vocabulary should be consistent, including that of printed documentation, online documentation, and messages.

In the same way, data formats for dollar amounts or dates should be the same throughout an application. The same formats that are used in messages should be used in data entry. Complex data, like author/title/publisher for a book, should be formatted the same every time they are displayed.

Consistency is very hard to achieve, and will not be achieved unless it is one of the designers' main goals. User interface designers must have a detailed set of rules for what the user interface will look like, usually in the form of a user interface design handbook. This handbook will describe screen layouts, data entry procedures, command structures, online help, etc.

Sometimes a user interface design handbook is made public. For example, Apple has a set of rules for the Macintosh (Apple Inc., 1987) that describes what a standard Macintosh application looks like. The result of having a set of design guidelines is that a new Macintosh program can often be used without reading its manual.

It would be very helpful if each software package would come with a copy of the user interface design handbook used to build it. For one thing, those systems without a handbook would be obvious and could be avoided. Unfortunately, most companies consider their handbook to be proprietary. The user manual should be able to provide a general description of the design rules which one can use to learn how to use the application and to determine whether a consistent set of rules was used to design the user interface. It is important to find out whether or not an application was designed with a user interface design handbook, since it is nearly impossible for a large system that was built by many people to have a consistent user interface without one.

#### *Hide Unnecessary Details*

Lurking beneath the surface of every application program is a hoard of details that are completely uninteresting to the user, not least of which are the computer's hardware and operating system. These details should be hidden from the user. The user should never see a message from the operating system complaining about bad file names or lack of disc space. Instead, the application should catch these errors itself or explain them in the user's own language. A user interface should communicate the purpose of the application, not the peculiarities of the system software that it is using.

Users do not care about how the application is implemented, either.

For example, the original catalog searching program at the University of Illinois at Urbana-Champaign forces users making title searches to type in the first four letters of the first significant word of the title and the first five letters of the second. This is presumably done because of the way that the database is organized, but users don't care about this detail. Users should be able to enter the entire title and let the computer figure out which words are important.

### **Take the User's Point of View**

There are many ways to characterize groups of users. An information retrieval system might be designed for professionals who plan to use it for an hour or more every day. These users are willing to spend several hours learning to use the system, but need an interface that makes good use of their time once they have mastered it. On the other extreme, the system might be designed for members of the general public who will not use it more than a few times a year. These users need an interface that leads them through the functions of the program without bewildering them. University students might make up a third group, since they will usually be familiar with computers, though not with the principles of information retrieval, and can use that knowledge to learn about new systems.

Some computer systems with a reputation for poor user interfaces actually have interfaces that are well designed for a small, atypical group of users. A good example is UNIX, which is an operating system developed by a few computer scientists within Bell Laboratories and which is favored by computer science departments. According to legend, the large number of cryptic two- and three-letter commands were chosen because the early designers were slow typists and wanted to be able to quickly enter a large number of commands. Thus, the same features that make UNIX hard to use by the general public made it well suited for its original set of users.

### *Adopt the User's Vocabulary*

The importance of speaking the user's language is most clear for software being shipped to another country. However, even among English speakers, different people have quite different vocabularies. In particular, software designers are usually quite different from users and have a different vocabulary. Thus, many programs use computer jargon to communicate with those who know little about computers. Just as bad is an accounting program that uses accountant's jargon to communicate with the new manager of a small business who is trying to learn to use it, or an online catalog that uses librarian's jargon to communicate with

patrons. Inappropriate jargon is worse than meaningless, because it can make the user feel stupid and frustrated.

Jargon is not necessarily bad. If a group of users share a common language, then programs that use that language will be able to communicate with them better. However, users outside that group will be confused by the jargon. For example, an expert system that is helping a doctor diagnose a disease must use the medical terms with which the doctor is familiar, but these terms will be meaningless to most other people.

### *Adopt the User's Assumptions*

It is easiest to communicate with someone with whom one shares a common set of knowledge and beliefs. The main advantage to basing computer systems on some preexisting paper system is that users will already know the kinds of information stored in the computer and what can be done with it. Thus, users will start off with a close approximation to the application model. This will decrease training time and anxiety level of new users.

The expertise of users can be measured along several dimensions, such as their knowledge of computers and their knowledge of the application domain. Patrons of a computer science library can be expected to be experts on operating a computer, but they will not know any more about how a library works than other library patrons.

Hidden assumptions often cause problems. For example, people have preconceived ideas about many colors. Auto drivers think of red as meaning stop or danger, chemical engineers think of red as meaning hot, and electrical engineers use red lights to mean that the electrical power is on. It is easy for users to place their own interpretation on the use of colors even when this goes against the intention of the user interface designer.

### *Understand the Needs and Characteristics of the User*

People rarely want to use a program just because of its fancy interface. Instead, they have a job to do and think that the program can help them do it. For example, most students use the library to write papers for classes. They usually know the subject of the paper, but not the books on that subject. The traditional card catalog, which puts author and title at the same priority as the subject, is not suited to their needs.

Typing skills can have a big impact on how easy a particular program is to use. Some programs almost eliminate typing by using menus and touch screens or mice. A skilled typist will find a touch

screen interface considerably slower than one based on a keyboard, but most other people will prefer to use a keyboard as little as possible.

Computer systems in libraries have many different kinds of users. Patrons of public libraries can be characterized by a general unfamiliarity with the intricacies of either libraries or computers, but there is little else they have in common. Some are poor typists, some are hard of hearing, some are color blind, some are quite young, and others are elderly. Patrons of a university library will be more homogeneous, but even in a university library the user interface for patrons is probably the biggest challenge for the user interface designer.

There are other important groups of users in a library. Professional librarians are likely to be willing to spend time to learn the details of a program, and might be interested in user interfaces that they can customize to make as efficient as possible. Data entry clerks are likely to be skilled typists but will not know much about computers. Some patrons, such as graduate students or others doing research, might be more concerned with the efficiency of the user interface and less concerned about its ease of use. It is hard for one interface to satisfy all these users.

## TRENDS IN USER-INTERFACE DESIGN

The last few years have seen an explosion in the amount of software developed to be easy to use. This is due in large part to the popularity of personal computers, which have introduced many new people to computers, making it more important for software to be easy to use and providing a large market for good user interfaces. This has resulted in several new ideas in user interface design. Some, like direct manipulation and customizable interfaces, are certain to be even more important in the future.

### Menus

A set of user interface guidelines published in 1975 did not even mention menus (Engel & Granda, 1975). Now they are almost synonymous with user-friendly software. Menus rely upon the fact that it is easier to recognize the name of a desired command than it is to remember it. Thus, menus are especially helpful for novice users. However, even experts will appreciate menus for those parts of a program that are rarely used.

Menus alone will not make a program easy to use. In fact, menus can be easily misused. Menus should not be too large or too small, they should be well formatted, menu items should have well-chosen names,

they should be given some kind of logical organization with spaces between the different categories, and there should be standard ways to exit a menu without selecting any of the choices. In general, if one has used menus that are implemented well, then poorly implemented menus will be obvious. If one did not design the menu, then poorly chosen menu names and poorly organized menus will confuse the user, but the designer of such menus never seems to be bothered by them.

There are several kinds of menus. One kind is a list of options labeled with letters. The user types in the label of an option to select it. These kinds of menus can be used on nearly any kind of terminal. Other menus are sets of icons or lists of words that can be selected by a pointing device. These kinds of menus require special graphics terminals with pointing devices. Although these kinds of menus seem very different, they are all effective at reducing memorization.

The major problem with menus is that they can slow down experts. Experts prefer user interfaces that allow them to issue commands from the keyboard without having to navigate through a set of menus. Unless the terminals are fast, displaying menus can take a lot of time. Thus, menu systems often allow users to also enter commands from the keyboard or to type a string of item names corresponding to a series of selections from the menus. These interfaces can be easy to use by novices and still efficient for experts.

### **Multiple Interfaces**

One obvious solution to the problem of having to cater to the needs of many kinds of users is to provide an interface for each kind. This was very hard to do in earlier applications because the user interface was implemented as an integral part of the application. A recent trend is to separate the construction of the user interface from that of the rest of the application. This is useful because user interfaces need to be revised more often than other parts of the application, because it makes it easier for the user interface to be designed by human factors experts, and because it makes it easier to provide several interfaces for a single application.

The Macintosh places all messages to the user in a special "resource" file. By providing a different resource file, all the messages in the application can be changed. This makes it easy to provide foreign language versions of applications or even to provide slightly different messages for different kinds of users.

Another way to provide multiple interfaces is to provide several different modes for different levels of user. For example, Microsoft Word has a "short menu" mode and a "long menu" mode. Short menus

contain only the most used commands, keeping new users from becoming confused by too many choices. Long names provide the experienced user with the entire set of commands.

### **Customizable Interfaces**

A customizable user interface is one that the user can change. Users might be able to change the name of menu items or function keys, to add new menu items or function keys that invoke some combination of existing commands, or to invent new commands. Different user interfaces vary widely in their customizability, but even the ability to change command names, menu names, or messages lets users with different vocabularies coexist.

Weissman discusses customizable user interfaces in depth in this proceedings. Customizable user interfaces are most valuable to expert users, since they can afford to spend the time to customize and often are more particular about it. Even novices can benefit from customizable interfaces, since an expert can easily modify the interface to be more suitable for a particular group of novice users.

### **Direct Manipulation**

A direct manipulation interface provides the illusion that the user is physically manipulating the data that is inside the computer. The model of the application becomes so real to the users that there is little translation from the user's intentions to the actions needed to carry out the intentions. Direct manipulation interfaces often make heavy use of graphics and pointing devices, letting the user select and move objects that appear on the computer screen.

When a direct manipulation interface is combined with an application model that the user already understands, it provides a very easy to use interface. Electronic spreadsheets and word processing programs are both good examples. Paper spreadsheets have long been used for analyzing budgets and for accounting. Electronic spreadsheets model the paper spreadsheets fairly directly except that they automatically calculate the spreadsheet according to rules entered by the user. Changing the numbers or calculation rules in an electronic spreadsheet causes an immediate change in the numbers that depend on them. This immediate feedback is an important part of direct manipulation systems.

Early spreadsheet programs like VISICALC and Lotus 1-2-3 used function keys to select and modify individual cells, which meant that it took a little time for a new user to figure out how to use these programs. Users of newer programs like Excel can select cells by pointing at them with a mouse and can modify them by typing in new values. The

graphical display makes the electronic version of the spreadsheet look exactly like the paper version, so new users have no problem figuring out what is going on. The improved interface makes manipulating the newer spreadsheets even more direct.

Word processing programs provide a direct manipulation interface when they display the document being created in a form as much like the eventual printed form as possible. Text is formatted as it is added, so the display always shows the current state of the document. Text is deleted or moved by selecting it with the pointing device. Many complicated operations, such as changing the size of characters or the fonts being used, are invoked by complicated techniques. However, the basic editing operations are always very simple and intuitive. Using these systems is like rearranging words in a magic book that not only formats the text automatically, but complains when words are misspelled.

A direct manipulation interface to the cataloging database could model the card catalogs with which patrons are familiar. Users could fill in part of a card and have the computer find all cards that match the information that was filled in. Users could move the images of interesting cards to one side of the screen, throw away the images of cards that were not interesting, copy information from interesting cards into blank cards to request more items from the catalog, and end up with sets of cards that they can print out or use to check out the corresponding documents.

## New Hardware

It is clear that graphics is becoming more and more important to personal computer users. New personal computers all have graphics capabilities, and the cheapest of these machines are about the same price as some character-only display terminals.

While color is important to people and will eventually help make user interfaces more effective, there are only a few special areas in which it is known how to make good use of color graphics. A color interface is obviously useful in preparing color charts or illustrations. Integrated circuit design and some other computer-aided design systems make effective use of color. However, many attempts to use color result in garish, hard-to-read displays.

There are a few ways in which color is useful in text-based applications. Color works well to attract the user's attention, especially to changes in status. It allows the user to spot related items that are widely scattered on the screen. It also can help format a densely packed display. However, color displays are more expensive than black and

white displays, and it is not at all obvious that color displays are cost effective in the library environment.

Pointing devices like mice or track-balls can be used in conjunction with graphics to reduce the use of the keyboard and make it possible for people to use the computer with no training at all. The need for memorization can be reduced to a minimum, and graphics can be used to provide excellent feedback. Thus, systems that combine simple graphics and pointing devices will become more common.

Touch screens have been available for a long time, but software technology has finally advanced to the point where they can be useful. Using a touch screen for a long time will make most people's arms tired, but the screens are easy and intuitive to use for novices. Thus, there will probably be a place in libraries for touch screens.

An interesting new idea that shows much potential is the data glove. This is a glove with sensors in it that the computer can use to determine the position of the user's hand. The computer can tell whether the hand in the glove is pointing, grabbing, or pushing items on the display. The result is a remarkably powerful interface. However, full development of this kind of interface is several years away.

Voice output is becoming more popular, showing up in videogames and personal computers. Voice input is far less capable. Some voice input systems can understand a small fixed vocabulary for a general population, or be trained for a large vocabulary for a single person, but none are even close to being suitable for use by library patrons.

## FINDING A GOOD USER INTERFACE

It is easiest to buy or build software for oneself. Since the buyer and the user are the same person, if one buys what one enjoys using, then one will usually make the right choice. However, no new user is an expert at using the program, so if speed for an expert is important, then one should not buy a program based on how one liked using it for an hour in the store.

Unfortunately, software is often bought or designed by someone other than its final user. People usually buy or design software that they would like to use, even when the main users of the software will be other people. Since a user interface that is good for one person might be bad for another, the result is frequently software that is poorly suited for its users.

Software for the personal computer mass market is reviewed extensively by many magazines. These reviewers usually examine closely how easy the application is to use and to learn. Inconsistencies in the

user interface, lack of documentation, or confusing commands will provoke negative comments from the reviewers. Moreover, since this kind of software is so widely used and has so much competition, it is relatively easy to find people who have used several competing systems and can provide an accurate description of their relative strengths and weaknesses. However, different users have different needs, and one may need customized software or software for less popular computers. Thus, it is important to know how to evaluate a user interface.

The amount of effort one puts into finding a system with a good user interface depends on the importance of making the right choice. An extensive evaluation of a program can be expensive and is only necessary if the software is expensive, will play a crucial role in one's organization, or will be used by lots of people. Often the cost of training and recovering from a mistaken purchase is much more than the cost of the purchase itself, so one should not underestimate the importance of making the right choice.

## Testing

Testing a user interface follows the methodology of the social sciences more than that of engineering. Perhaps this is one of the reasons why so many software systems have poor user interfaces. Their designers often have an engineering background and have not been trained to analyze human behavior.

First, a representative sample of users must be gathered. Only a few users of each type are needed, but each kind of user must be represented. The exact number of people depends on the difficulty of performing the tests and the importance of making the right choice. Finding the right set of users is probably the hardest part of testing a user interface, and requires a careful analysis of the user population and how they will use the system.

In place must be a defined list of representative tasks for the users to perform. Groups of people that differ in the reasons they want to use the system or in the functions they will perform with it will need different lists of tasks.

Given a set of users and a list of tasks, the quality of a user interface can be measured by the amount of time it takes each user to successfully complete his or her list of tasks. Although there are other properties that could be measured besides the speed of using the system, it turns out that most of them are included in measuring speed. For example, if an interface causes users to make many errors, a lot of time will be spent making corrections, and it will take longer to complete a set of

tasks. Speed is a poor measure if many users never complete their tasks as is discussed by Leslie Edmonds in this proceedings.

It can be hard to evaluate a user interface designed for experts. First, it takes time to train users to be experts so that the test is valid. Second, experts will probably use the computer for hours at a time, so ergonomic properties of the user interface are important for them and need to be evaluated. Third, systems used by experts are usually more complex, require more documentation, and can make use of features such as customization of the user interface that are not as useful for novices. Thus, expert users still have a much larger list of tasks than novices. It takes more time to evaluate a user interface for experts than one for novices.

This form of evaluation is not adequate for system designers, who need to diagnose problems with a user interface so that problems can be corrected. However, it works well for deciding which existing system is easiest to use.

## Questions

There are a number of questions that help to evaluate the user interface of an application. Many of these questions cannot have any absolute answer, so it is best to use them to compare competing systems. The following list is probably too short, but it provides a good indication of the kinds of questions that are important.

- For whom is the system designed?
- Is the system too complex?
- Is the basic idea well-known to the intended users?
- Is there enough feedback?
- Is there high quality online documentation?
- Is the user forced to memorize commands? Is the user forced to flip from one screen to another?
- Is the system consistent?
- Is there a set of rules to ensure consistency?
- Are implementation details hidden from the user?
- Is the vocabulary of the program the same as that of the users?
- Are menus and displays well organized?
- Are command names easy to understand?

Answering these questions can help one understand why a particular system is hard to use and will be helpful in persuading the designers to fix their mistakes. However, the answers will be subjective and are no substitute for objective measurements. Good user interfaces are important because they make the users more productive, so measuring the productivity of users is the ultimate test of a user interface.

## CONCLUSION

User interface design has improved significantly in the last few years. However, designing a good user interface is still difficult and expensive. Software vendors are not likely to design good user interfaces unless their customers demand it. If expectations of software quality are increased, then software vendors will be forced to provide user interfaces that are efficient and easy to use.

## REFERENCES

- Apple, Inc. (1987). *Human interface guidelines*. Reading, MA: Addison-Wesley.
- Dumas, J. S. (1988). *Designing user interfaces for software*. Englewood Cliffs, NJ: Prentice Hall.
- Engel, S., & Granda, R. (1975). *Guidelines for man/display interfaces*. IBM Technical Report TR00.2720. Yorktown Heights, NY: IBM Research Division, T. J. Watson Research Center.
- Heckel, P. (1984). *The elements of friendly software design*. New York: Warner Books.
- Meyrowitz, N. (1986). Intermedia: The architecture and construction of an object-oriented hypermedia system and application framework. *SIGPLAN Notices*, 21(11), 186-201.
- Norman, D. A. (1981). The trouble with UNIX. *Datamation*, 27(12), 139-150.
- Schneiderman, B. (1987). *Designing the user interface: Strategies for effective human-computer interaction*. Reading, MA: Addison-Wesley.